УДК 004
DOI: 10.21122/2309-4923-2024-2-4-10

KARASIK O.N., PRIHOZHY A.A.

BLOCKED ALGORITHM OF FINDING ALL-PAIRS SHORTEST PATHS IN GRAPHS DIVIDED INTO WEAKLY CONNECTED CLUSTERS

Belarusian National Technical University
Minsk, Republic of Belarus

The problem of finding all shortest paths between vertices in a graph (APSP) has real-life applications in planning, communication, economics and many other areas. APSP problem can be solved using various algorithms, starting from Floyd-Warshall's algorithm and ending with advanced, much faster blocked algorithms like Heterogeneous Blocked All-Pairs Shortest Path Algorithm designed to fully utilize underlying hardware resources and utilize inter-data relationships. In the paper, we propose a novel Blocked all-pairs Shortest Paths algorithm for Clustered Graphs (BSPCG) (in sequential and parallel forms) which utilizes the graph clustering information to significantly reduce the number of calculations by performing shortest paths search only though bridge vertices between clusters. We performed a set of comparing experiments for BSPCG and standard Blocked All-Pairs Shortest Path (BFW) algorithm on four randomly generated graphs of 4800 and 9600 vertices with different cluster configurations to determine the efficiency of calculation of paths passing through bridge vertices. All experiments were executed on a computer with two Intel Xeon E5-2620v4 processors (8 cores, 16 hardware threads and shared 20 MB L3 cache). In all the experiments the novel BSPCG algorithm outperformed the standard BFW algorithm. In single-threaded scenarios, BSPCG outperformed BFW up to 4.6 times on graphs of 4800 vertices and up to 2.7 times on graphs of 9600 vertices. In the multi-threaded scenarios, BSPCG also outperformed BFW up to 4.0 times on graphs of 4800 vertices and up to 2.7 times on graphs of 9600 vertices. The proposed algorithm can be used in scenarios where clustering information stays intact or slightly modified based on the changes in graph and can be reused for future calculation of all-pairs shortest paths in the graph.

Keywords: *shortest paths algorithm, blocked algorithm, graph clustering, single-thread application, multi-threaded application, speedup*

Introduction

The problem of finding shortest paths between vertices in a graph has multiple real-life applications. It is used to solve mazes, optimize traffic networks, to improve task planning, etc [1–3]. The problem can be formulated as to find shortest paths originated from one source (Single Source Shortest Path – SSSP) and to find shortest paths between all-pairs of vertices (All Pairs Shortest Path – APSP). Dijkstra's algorithm [4] is a classic solution to SSSP and Floyd-Warshall's algorithm [5] is a classic solution to APSP.

In context of the APSP problem, the Floyd-Warshall's algorithm demonstrate maximum efficiency when it is applied to dense or complete graphs and much less efficiency when it is applied to sparse graphs. The same rule applies to algorithm's modifications, which are primarily focused on improving memory usage and effective parallelization [6–9].

In our recent research [10–12] we started to focus on applicability of a Blocked Floyd-Warshall algorithm (BFW) to sparse graphs and on utilization of additional information about the graph to reduce the calculation time and resource consumption. In this paper, we focus on extension of the recently proposed Blocked Shortest Paths algorithm [11] with Unequally Sized blocks (BSPUS) using information about graph clustering to significantly reduce the number of calculations on sparse graphs with weakly connected clusters.

APSP algorithms

Let a directed graph G consists of a set V of vertices (numbered $1…N$) and a set E of edges with real edge-weights. A cost adjacency matrix D of size $N×N$ represents G. It is initialized with weights of the edges in such a way, that element $D[i, j]$ is the weight of the edge between vertices i and j.

Floyd-Warshall's (*FW*) algorithm [5] iterates over the cost adjacency matrix D and checks existence of a path from vertex i to vertex j through existence of paths from i to k and from k to j (Figure 1). The algorithm always iterates over all the vertices and doesn't account for the density or structure of the graph.

Blocked Floyd-Warshall's (*BFW*) algorithm [8] operates on a matrix of blocks B which is created by dividing a matrix D into M equally sized blocks of size L in such a way that $M*L=N$ (Figure 2).

The outer loop of BFW has M iterations. Each of them performs (Figure 3):

1. Calculation of "diagonal" block.
2. Calculation of "cross" blocks.
3. Calculation of "peripheral" blocks.

```
function FW() {
  for k ∈ {1 … N} {
    for i ∈ {1 … N} {
      for j ∈ {1 … N} {
        D[i,j] = min(D[i,j], D[i,k] + D[k,j])
      }
    }
  }
}
```

Figure 1. Pseudocode of original Floyd-Warshall (FW) algorithm

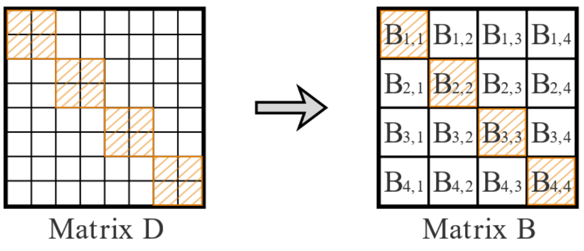


Figure 2. Visualization of matrix D being split into matrix of blocks B

All calculations are performed by a single procedure which accepts three input blocks at a time. The procedure calculates the paths between all vertices represented by block $B1$ passing through the vertices represented by $B2$ and $B3$ (Figure 4). In scope of one iteration, calculations have the following effect:

● In case of diagonal block, the procedure accepts the single diagonal block as $B1$, $B2$ and $B3$. This results in calculation of all shortest paths between pairs of vertices associated with the diagonal block.

● In case of cross blocks, the procedure accepts the diagonal block as $B3$ (vertical) or $B2$ (horizontal) and the cross block as $B1$ and $B2$ or $B1$ and $B3$ respectively. This results in a calculation of all shortest paths between all vertices of horizontal and vertical blocks of cross through vertices of the diagonal block.

● In case of peripheral blocks, the procedure accepts one peripheral and two cross blocks (vertical and horizontal) as $B1$, $B2$ and $B3$ respectively. This results in calculation of all shortest paths between pairs of vertices of peripheral block through vertices of diagonal block.

It should be noted that in each iteration BFW calculates the shortest paths between all vertices through vertices of the diagonal block.

Blocked all-pairs Shortest Paths algorithm with Unequally Sized blocks (*BSPUS*) was proposed in [11]; it generalizes the idea of *BFW* for graphs divided into unequally sized subgraphs. It extends the capabilities of existing blocked *APSP* algorithms and allows to solve the *APSP* problem on graphs that are partitioned into weakly connected dense clusters. The algorithm operates on a matrix U of blocks, which is created by dividing matrix D into M unequal blocks of sizes $S = \{S_1 \dots c\}$ in such a way that $S_1+\dots+S_M = 1$. (Figure 5).

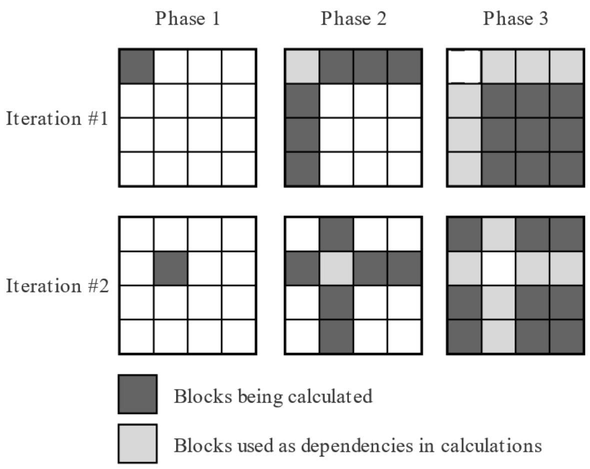


Figure 3. Visualization of two frist iterations of Blocked Floyd-Warshall (BFW) algorithm

```
function BFW() {
  for m ∈ {1 … M} {
    proc(B[m,m], B[m,m], B[m,m]);
    for i ∈ {1 … M} and i ≠ m {
        proc(B[i,m], B[i,m], B[m,m]);
        proc(B[m,i], B[m,m], B[m,i]);
    }
    for i ∈ {1 … M} and i ≠ m {
        for j ∈ {1 … M} and j ≠ m {
            proc(B[i,j], B[i,m], B[m,j]);
        }
    }
  }
}
```

```
function proc(B1, B2, B3) {
  for k ∈ {1 … L} {
    for i ∈ {1 … L} {
      for j ∈ {1 … L} {
        B1[i,j] = min(B1[i,j], B2[i,k] + B3[k,j])
      }
    }
  }
}
```

Figure 4. Pseudocode of Blocked Floyd-Warshall (BFW) algorithm

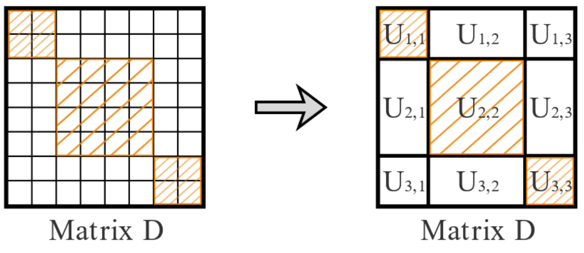


Figure 5. Illustration of matrix D being split into matrix U of blocks with unequal sizes

The block calculation procedure Figure 6 in *BSPUS* differs from those in *BFW* by considering the parameters such as height (the number of rows) and width (the number of columns) of blocks.

```
function proc(U1, U2, U3) {
    for k ∈ {1 … height(U3)} {
        for i ∈ {1 … height(U1)} {
            for j = ∈ {1 … width(U1)} {
                U1[i,j] = min(U1[i,j], U2[i,k] + U3[k,j])
            }
        }
    }
}
```

Figure 6. Pseudocodes of block calculation procedure in BSPUS

Analysis of algorithms. All presented algorithms have the same computational complexity – $O(N^3)$ and space complexity – $O(N^2)$. However, they have different performance and energy consumption characteristics [13–16]. *BFW* is more advanced than *FW* regarding both parameters because of the improved memory access pattern and spatial data locality. *BFW* also provides multiple opportunities for performance optimizations by using different calculation procedures for different blocks [7] and effective parallelization using blocks interdependencies [17].

In addition to benefits of *BFW*, the *BSPUS* provides an opportunity to adapt algorithms execution to the graph structure by splitting it into blocks based on the graph clusters [12]. In one of our previous works [10] we defined requirements for methods of graph clustering and demonstrated that existing, well-known graph clustering methods can fulfil these requirement given a sparse graph.

In this paper, we extend the application of graph clustering results to improve the methods of solving the all-pairs shortest path problem. We extend the *BSPUS* algorithm to a *Blocked all-pairs Shortest Paths algorithm for Clustered Graphs* (*BSPCG*), which uses the graph clustering information to reduce the number of calculations significantly regarding sparse graphs.

Blocked all-pairs Shortest Paths algorithm for Clustered Graphs

Figure 7 shows a small example graph G of 16 vertices interconnected by 28 edges. Visually it is easy to spot (Figure 8) three highly interconnected clusters C_1, C_2 and C_3 where:

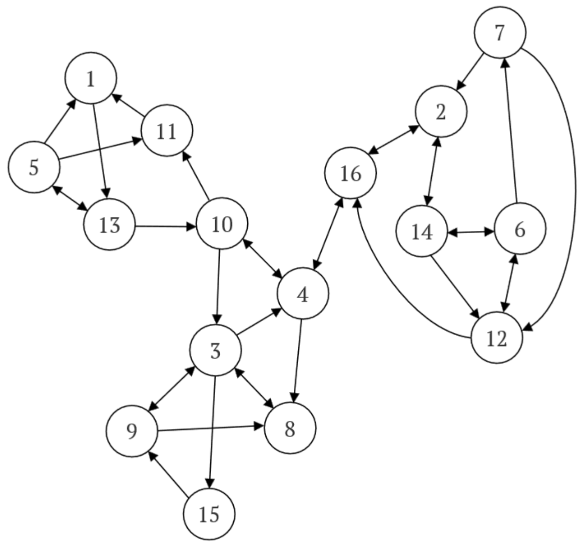


Figure 7. Example graph G of 16 vertices and 28 edges

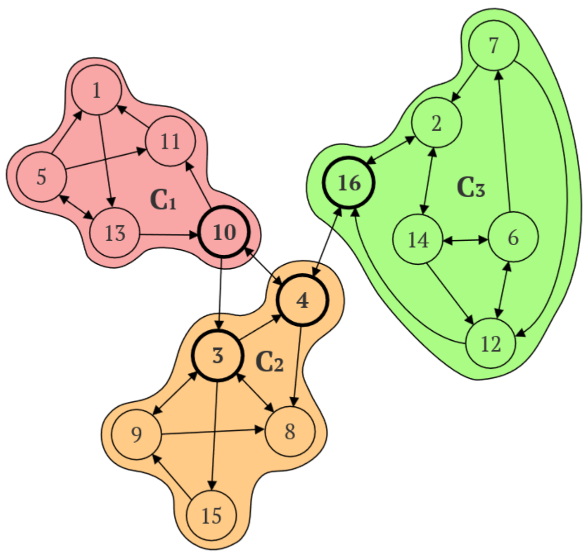


Figure 8. Example graph G split into three clusters C_1 (red), C_2 (orange) and C_3 (green) interconnected through 4 bridge vertices – 3, 4, 10 and 16 (in bold)

- C_1 includes vertices 1, 5, 10, 11 and 13.
- C_2 includes vertices 3, 4, 8, 9 and 15.
- C_3 includes vertices 2, 6, 7, 12 and 16.

These clusters are interconnected by 4 bridge vertices – 3, 4 (from C_2), 10 (from C_1) and 16 (from C_3), and 5 directed edges – 10 → 3, 4 → 10, 10 → 4, 4 → 16 and 16 → 4.

Looking at Figure 8 it can be observed that if we calculate all shortest paths from vertex 4 to all vertices of cluster C_1, then all these paths would lay straight through vertex 10 (which is a bridge vertex of C_1). Absolutely the same is true, if we would like to calculate all shortest paths from any vertices of C_2 or C_3 to vertices of C_1. It remains true, if we calculate all shortest paths from vertices 1, 5, 11 or 13 into vertices of C_2 or C_3. In context of the *FW*, *BFW* and *BSPUS* algorithms it means that when we calculate such paths we don't need to iterate over all vertices of the graph, instead, we only need to iterate through cluster's bridge vertices – in our case, vertex 10.

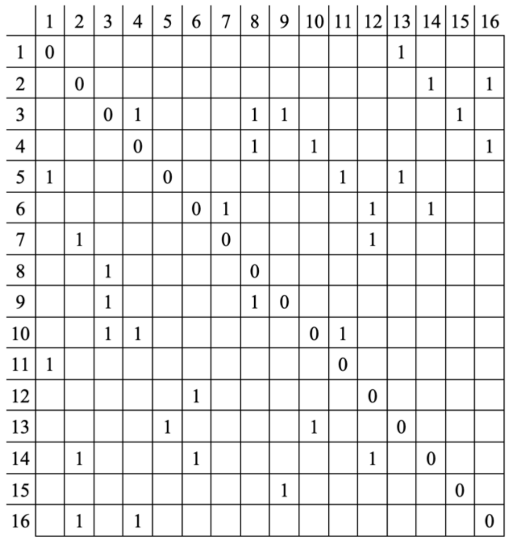
However, before calculating a path to or from the cluster through its bridge vertices we must first calculate all shortest paths within the cluster.

The *BSPCG* algorithm utilizes the opportunity to use unequally sized blocks (provided by *BSPUS*) and iterative mechanics of the *BFW*, specifically the part where in every iteration the diagonal block is calculated first, which results into calculations of all shortest paths between its vertices.

The algorithm we propose is as follows:

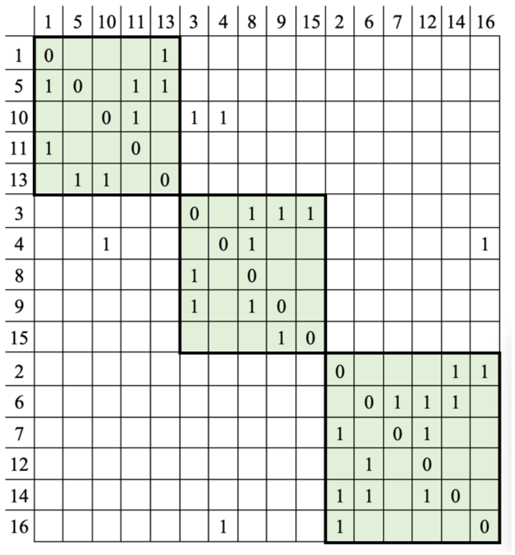
1. Represent a graph as cost-adjacency matrix (Figure 9*a*).
2. Rearrange its rows and columns to group clusters around matrix diagonal. Split the matrix into unequally sized blocks where clusters are diagonal blocks (Figure 9*b*).
3. Calculate vectors *I* of block relative indices of all bridge vertices for every diagonal block (Figure 10).
4. Calculate all-pairs shortest paths using *BSPCG* algorithm (Figure 11).
5. Rearrange rows and columns to return them to their original positions.

In step #2, the algorithm incorporates graph clusters into the matrix and makes it compatible with the iterative mechanics of the *BFW* and *BSPUS* algorithms (Figure 3).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0												1			
2		0												1		1
3			0	1				1	1						1	
4				0				1		1						1
5	1				0						1		1			
6						0	1					1		1		
7		1					0					1				
8			1					0								
9			1					1	0							
10			1	1						0	1					
11	1										0					
12						1						0				
13					1					1			0			
14		1				1						1		0		
15									1						0	
16		1		1												0

a

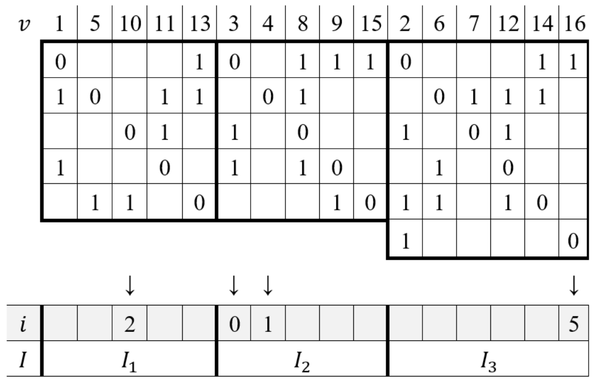


	1	5	10	11	13	3	4	8	9	15	2	6	7	12	14	16
1	0				1											
5	1	0		1	1											
10			0	1		1	1									
11	1			0												
13		1	1		0											
3						0		1	1	1						
4			1				0	1								1
8						1		0								
9						1		1	0							
15									1	0						
2											0				1	1
6												0	1	1	1	
7											1		0	1		
12												1		0		
14											1	1		1	0	
16							1				1					0

b

Figure 9. Representation of example graph *G* with cost-adjacency matrix:
a) is original representations and *b*) is a cost-adjacency matrix where rows and colums are rearranged to group clusters across matrix diagonal, splitted into 9 unequally sized blocks (one diagonal block for each cluster)

In step #3, the algorithm transforms information about bridge vertices into the format compatible with the algorithm structure where k is not the vertex number in a graph but a block relative index of it.



v	1	5	10	11	13	3	4	8	9	15	2	6	7	12	14	16
	0				1	0		1	1	1	0				1	1
	1	0		1	1		0	1				0	1	1	1	
			0	1		1		0			1		0	1		
	1			0		1		1	0			1		0		
		1	1		0				1	0	1	1		1	0	
											1					0
			↓			↓	↓									↓
i			2			0	1									5
I			I_1					I_2						I_3		

Figure 10. Illustration of mapping diagonal block verticies (v) of graph G to their block relative indices (i) in index vector I

In step #5, the algorithm reverses the changes made in step #2 and returns the structure of the matrix to its initial state.

```
function $BSPCG()$ {
  for $m \in \{1 \dots M\}$ {
    $proc(B[m,m], B[m,m], B[m,m])$;
    for $i \in \{1 \dots M\}$ and $i \neq m$ {
        $proc\_bridges(B[i,m], B[i,m], B[m,m], I(m))$;
        $proc\_bridges(B[m,i], B[m,m], B[m,i], I(m))$;
    }
    for $i \in \{1 \dots M\}$ and $i \neq m$ {
      for $j \in \{1 \dots M\}$ and $j \neq m$ {
        $proc\_bridges\ (B[i,j], B[i,m], B[m,j]\ , I(m))$;
      }
    }
  }
}
```

Figure 11. Pseudocode of BSPCG algorithm with two calculation procedures – proc to calculate diagonal blocks and proc_bridges to calculate cross and peripheral blocks

In terms of algorithmic complexity, *BSPCG* has the same space and worst-case time complexity as *FW*, *BFW* and *BSPUS*. However, it should be noticed that the number of computations in *BSPCG* depends on the number of bridge vertices between clusters and on the overall graph size. Therefore, in practical cases, the usage of *BSPCG* on sparse graphs can result in a significant speedup in computations.

Experiments

Environment. We implemented sequential and parallel versions of the *BFW* and *BSPCG* algorithms using C++ language. The parallel version was implemented using OpenMP v4.5. The source code was compiled by GNU GCC compiler v13.2.0 with auto-vectorization enabled.

Experiments were run on a computer with two Intel Xeon E5-2620v4 processors (8 cores, 16 hardware threads, L1 cache 32 KB, L2 cache 256 KB) with shared inclusive 20 MB L3 cache.

Graphs. We generated four random connected sparse graphs with predefined cluster configurations (Table 1).

The sizes of clusters in each of the experimental graphs have been varied in a certain range, and the share of bridge vertices in the overall number of vertices was not high.

Table 1. Specification of experimental graphs that are randomly generated for a predefined number of clusters

Graph	1	2	3	4
Vertices	4800	4800	9600	9600
Edges	288245	153858	644198	326779
Clusters	20	41	40	80
Bridge vertices	567	620	3452	3550
Bridge edges	621	687	2374	2505

We intentionally generated pairs of graphs split into different number of clusters but with close number of bridge vertices to evaluate the influence of our above formulated assumption on the *BSPCG* complexity.

Configuration. In all experiments we set block size of *BFW* algorithm to 120×120. This size was found to give the best results on the selected experimental system [13]. In *BSPCG* the sizes of the blocks were derived from the sizes of the clusters.

Results. We performed a set of experiments, each repeated at least five times. The standard deviation of the running time has not exceeded the 0.8 % mark. All results for single- and multi-threaded implementations are presented in Table 2.

Table 2. Experimental results of single and multi-threaded implementations of the *BFW* and *BSPCG* algorithms on 4 experimental graphs of 4800 and 9600 vertices. Time in presented in seconds (*s*)

Graph	4800		9600		4800		9600	
Clusters	20	40	40	80	20	40	40	80
Type	Single Thread				Multi-Threaded			
BFW	38.1	38.4	305.4	305.1	3.2	3.2	25.1	25.1
BSPCG	8.2	8.7	110.5	115.6	0.8	1.0	9.1	10.4

In the single-thread implementation, *BSPCG* outperforms *BFW* by 4.4 to 4.6 times on the graphs of 4800 vertices and by 2.6 to 2.7 times on the graphs of 9600 vertices. In the multi-threaded implementation, *BSPCG* outperforms *BFW* by 3.2 to 4 times on the graphs of 4800 vertices and by 2.4 to 2.7 times on the graphs of 9600 vertices.

We can observe from the Table 2 that the execution time of *BFW* doesn't depend on the number or clusters, nor the number of bridge vertices, nor the overall number of edges and, in general, isn't affected by graph structure but only by the size of the graph. Regarding *BSPCG*, it is easy to see that the execution time depends on both the graph size and the bridge vertex count.

Analysis. One of the potential disadvantages of *BSPCG* is the time required to retrieve graph clustering information. The potential impact of this depends on the algorithm used to calculate graph clustering information and can be nullified if such information exists prior to the execution.

The major advantage of the algorithm is its ability to reuse the same clustering information to recalculate shortest paths in the same graph but with different edge weights (a scenario for an optimi-zation problem). Besides reusing the same clustering information, it is also applicable in scenarios when incremental changes are made to the graph (eviction or addition of the edge) which can be reflected on the previously collected clustering information which then can be used to recalculate shortest paths.

Conclusion

In this paper we have presented a novel Blocked all-pairs Shortest Paths algorithm for Clustered Graphs (*BSPCG*) which can efficiently utilize information of graph clustering and significantly speedup the computation of all-pairs shortest paths in large sparse graphs. We implemented the *BFW* and *BSPCG* algorithms in single- and multi-threaded scenarios. We performed a set of experiments using four randomly generated sparse graphs of different sizes and different cluster configurations where the *BSPCG* algorithm outperformed the BFW algorithm on all experimental graphs by 2.4 to 4.6 times in both scenarios. The *BSPCG* algorithm opens a possibility for two potential use cases where clustering information is not recalculated each time before the calculation of shortest paths but instead is either maintained or slightly modified based on changes in the graph.

REFERENCES

1. **Schrijver A.** On the history of the shortest path problem. Documenta Mathematica. 2012. Vol. 17, № 1. P. 155–167.
2. **Anu P.**, (Kumar) M.G. Finding All-Pairs Shortest Path for a Large-Scale Transportation Network Using Parallel Floyd-Warshall and Parallel Dijkstra Algorithms. Journal of Computing in Civil Engineering. 2013. Vol. 27, № 3. P. 263–273.
3. **Ridi L., Torrini J., Vicario E.** Developing a Scheduler with Difference-Bound Matrices and the Floyd-Warshall Algorithm. IEEE Software. 2012. Vol. 29, № 1. P. 76–83.
4. **E.W. Dijkstra.** A note on two problems in connexion with graphs. Numerische Mathematik. 1959. Vol. 1, № 1. P. 269–271.
5. **Floyd R.W.** Algorithm 97: Shortest Path. Communications of the ACM. 1962. Vol. 5, № 6. P. 345-.
6. **Prihozhy A., Karasik O.** Inference of shortest path algorithms with spatial and temporal locality for Big Data processing. Proceedings VIII International conference “Big data and advanced analytics”, Minsk: Bestprint, 2022. P. 56-66.
7. **Prihozhy A., Karasik O.** Heterogenious blocked all-pairs shortest paths algorithm. System analysis and Applied Information Science. 2017. № 3. P. 68–75.
8. **Venkataraman G., Sahni S., Mukhopadhyaya S.** A Blocked All-Pairs Shortest Paths Algorithm. Journal of Experimental Algorithmics (JEA). 2003. Vol. 8. P. 857–874.
9. **Singh A., Mishra P.K.** Performance Analysis of Floyd Warshall Algorithm vs Rectangular Algorithm. International Journal of Computer Applications. 2014. Vol. 107, № 16. P. 23–27.
10. **Karasik O., Prihozhy A.** Requirements to methods of graph clustering at the aim of solving the shortest path problem. Proceedings X International conference “Big data and advanced analytics”, Minsk: BSUIR, 2024. P. 272–279.
11. **Prihozhy A., Karasik O.** New blocked all-pairs shortest paths algorithms operating on blocks of unequal sizes. System analysis and applied information science. BNTU, 2023. № 4. P. 4–13.
12. **Prihozhy A., Karasik O.** Blocked algorithm of shortest paths search in sparse graphs partitioned into unequally sized clusters. Proceedings X International conference “Big data and advanced analytics”, Minsk: BSUIR, 2024. P. 262–271.
13. **Karasik O., Prihozhy A.** Tuning block-parallel all-pairs shortest path algorithm for efficient multi-core implementation. System analysis and applied information science. BNTU, 2022. № 3. P. 57–65.
14. **Karasik O., Prihozhy A.** Parallel blocked all-pair shortest path algorithm: block size effect on cache operation in multi-core system. Proceedings VIII International conference “Big data and advanced analytics”, Minsk: Bestprint, 2022. P. 28–38.
15. **Prihozhy A., Karasik O.** Influence of shortest path algorithms on energy consumption of multi-core processors. System analysis and applied information science. BNTU, 2023. № 2. P. 4–12.
16. **Karasik O., Prihozhy A.** Profiling of energy consumption by algorithms of shortest paths search in large dense graphs. Proceedings IX International conference “Big data and advanced analytics”, Minsk: BSUIR, 2023. P. 44–50.
17. **Karasik O., Prihozhy A.** Streaming block-parallel algorithm for finding shortest paths on a graph. Minsk: BSUIR 2018. № 2. P. 77–84.

КАРАСИК О.Н., ПРИХОЖИЙ А.А.

БЛОЧНЫЙ АЛГОРИТМ ПОИСКА КРАТЧАЙШИХ ПУТЕЙ МЕЖДУ ВСЕМИ ПАРАМИ ВЕРШИН В ГРАФАХ СО СЛАБО-СВЯЗАННЫМИ КЛАСТЕРАМИ

Белорусский национальный технический университет
г. Минск, Республика Беларусь

Задача поиска кратчайших путей между всеми парами вершин в графе (APSP) имеет применяется в планировании, коммуникациях, экономике и многих других сферах. На сегодняшний день существует ряд алгоритмов решения APSP задач, начиная с алгоритма Флойда-Уоршелла (Floyd-Warshall) и заканчивая более продвинутыми и быстрыми блочными алгоритмами (например, неоднородным блочным алгоритмом поиска кратчайших путей - Heterogeneous Blocked All-Pairs Shortest Paths), предназначенными для максимально эффективного использования вычислительных средств и зависимостей между данными, участвующими в вычислениях. В статье предлагается новый блочный алгоритм BSPCG поиска кратчайших путей в кластеризованных графах в однопоточном и многопоточном вариантах, который использует информацию о кластеризации для сокращения объема вычислений посредством поиска кратчайших путей, проходящих через граничные вершины кластеров. В статье проведена серия вычислительных экспериментов над стандартным блочным алгоритмом BFW и новым алгоритмом BSPCG с целью доказательства эффективности поиска кратчайших путей в случае использования граничных вершин кластеров. Эксперименты выполнялись с использованием графов размером 4800 и 9600 вершин с различными кластерными конфигурациями. Эксперименты проведены на компьютере с двумя процессорами Intel Xeon E5-2620v4 (каждый процессор включает 8 физических ядер и 16 аппаратных потоков, а также кэш L3 объемом 20 МБ). Во всех проведенных экспериментах новый алгоритм BSPCG превзошел стандартный алгоритм BFW в несколько раз. В однопоточных сценариях BSPCG продемонстрировал ускорение по сравнению с BFW до 4.6 раз на графах с 4800 вершинами и до 2.7 раз на графах с 9600 вершинами. В многопоточных сценариях BSPCG также продемонстрировал ускорение до 4 раз на графах с 4800 вершинами и до 2,7 раз на графах с 9600 вершинами. Предложенный в статье алгоритм может быть использован в сценариях, где информация о кластеризации остается неизменной или изменяется незначительно и может быть повторно использована для множественных нахождений всех кратчайших путей в графе.

Ключевые слова: *поиск кратчайших путей на графе, блочный алгоритм, кластеризация графа, однопоточное приложение, многопоточное приложение, производительность*

Karasik Oleg is a Technology Lead at ISsoft Solutions (part of Coherent Solutions) in Minsk, Belarus, and PhD in Technical Science. His research interests include parallel multithreaded applications and the parallelization for multicore and multiprocessor systems.

Карасик О.Н., ведущий инженер иностранного производственного унитарного предприятия «ИССОФТ СОЛЮШЕНЗ» (часть Coherent Solutions), г. Минск, Беларусь, к.т.н. (2019). В сферу его научных интересов входят параллельные многопоточные приложения и распараллеливание для многоядерных и многопроцессорных систем.

E-mail: karasik.oleg.nikolaevich@gmail.com

Anatoly Prihozhy is full professor at Computer and system software department of Belarus national technical university, Doctor of Science (1999) and Full Professor (2001). His research interests include programming and hardware description languages, parallelizing compilers, and computer aided design techniques and tools for software and hardware at logic, high and system levels, and for incompletely specified logical systems. He has over 300 publications in Eastern and Western Europe, USA and Canada. Such worldwide publishers as IEEE, Springer, Kluwer Academic Publishers, World Scientific and others have published his works.

А.А. Прихожий, профессор кафедры «Программное обеспечение информационных систем и технологий» Белорусского национального технического университета, д.т.н. (1999), диплом профессора (2001). В сферу его научных интересов входят языки программирования и описания аппаратуры, распараллеливающие компиляторы, методы и средства автоматизированного проектирования программных и аппаратных средств на логическом, высоком и системном уровнях, а также не полностью определенных логических систем. Имеет более 300 публикаций в Восточной и Западной Европе, США и Канаде. Его работы опубликованы в таких мировых издательствах, как IEEE, Springer, Kluwer Academic Publishers, World Scientific и других.

E-mail: prihozhy@bntu.by